AMEMDMENTS TO THE SPECIFICATION

1. Please replace the title to read as follows:

On-Chip Debugging System With Shared Data Path and Sustained Evidence

2. Replace the following paragraphs as published with the following like-numbered paragraphs:

[0001] This application claims priority to U.S. provisional patent application No. <u>60/490,180</u> ~~60/______~~ , filed Jul. 25, 2003, entitled "Bug Tracer: Better Tool & Method to Diagnosis Software Bugs", which is incorporated herein by reference.

[0034] FIG. 1 is a <u>logical</u> block diagram of a debugging system in accordance with the present invention;

[0044] The central processor 14 also has control and instruction logic 15 which includes a reorder buffer 21. The reorder buffer 21 holds detailed description information regarding the operation of a program operating on the central processor. It will be appreciated that other processor architectures will have other low-level assets that may provide analogous detailed descriptions. When the failing program is executed on the central processor 14, the reorder buffer 21 holds detailed information regarding variables, branching, and other execution details. In order to use this valuable information, the reorder buffer 21 connects to a sentry circuit 31 and a reporter circuit <u>33</u> ~~31~~. Both the sentry circuit 31 and the reporter circuit 33 are constructed on-chip from fast-response circuitry, so are able to operate at or near the full processor speed. In one example, the sentry and reporter circuits include high-speed registers and comparators. However, it will be appreciated that other circuits may be used to implement such fast-response circuitry.

[0051] Referring now to FIG. 4, a debugging system 100 is illustrated. The debugging system 100 may be used by a software developer or other user to systematically locate the cause of software failures. Suppose the user has a central-program that can execute on a computer but fails to operate correctly. The failure includes at least one symptom that is known to the user, which indicates a bug (error) in the central-program. The user needs to diagnose this bug. For this task, the debugger 100 uses a systematic method and associated tools, particularly tools closely coupled to the processor 102 ~~101~~ of the computer. The debugger 100 provides several deep-level tools, which will be referred to as the "sentry" 110, "reporter" 111, "flags" 113, as well as an evidence buffer 136 in L2 cache 106. Further outboard or off-chip, this debugging system also has an evidence buffer 137 in main-memory 108, and an evidence file 138 in Storage 109.

[0069] The debugging system provides a tool (preparation software) to help the operator in translating this pseudo-code into specifications and programs for the sentry and reporter. During execution, the central-processor provides the sentry with a detailed description of each instruction. These include the instruction op-code (verb), data address, data value, instruction address. The sentry has means to compare this detailed description with specified op-codes (verbs), addresses, data values. The sentry includes serial or sequential logic (example: a co-processor). When a landmark occurs, the sentry may execute a specified corresponding action. For example, the sentry may assign a value to a landmark flag and activate the reporter. Then the reporter tests the landmark flag and branches to the corresponding action. This invokes reporter methods to extract and save evidence. The sentry and reporter are supported with "deep level resources". In many cases, these are highly compatible with the central-program. These can operate almost simultaneously with execution of the central-program, without major time-distortions. The operator programs or configures the sentry to detect each landmark during execution. Whenever the instruction op-code (verb) is "write to memory", the sentry compares the data address against a specified address, and compares the data value against a specified value. Also, the operator programs the

sentry and reporter with corresponding action for each landmark. With these preparations, the central-program is again executed. This extracts and saves a new evidence file that corresponds to the above hypothesis. The operator can use the exhibitor and detective software tools to examine this evidence file. Thus the operator systematically and efficiently can extract and save evidence that relates to the following diagnostic questions: "During Execution of Central-Program, after occurrence of this Precursor, when did this Symptom occur? How and why did this Symptom occur?" This may be abbreviated as follows: "When, how and why did this Symptom occur?" To help the operator examine the evidence file, the detective software provides efficient means to ask and answer a similar diagnostic question. Thus an operator can proceed from a precursor and symptom, to extract and save evidence that clarifies these. This debugging process can be iterated. Thus in many cases, an operator can efficiently and systematically extract and follow evidence from a symptom to the bug that caused it.

[0108] More Advanced Example: Suppose the operator hypothesizes there are three categories of methods: (i) ~~(ii)~~ new code that might contain a bug, where more detailed evidence is justified; (ii) new code that probably is less related to the bug, where less detailed evidence is justified; or (iii) mature code that probably is bug-free, where evidence is not justified. By using three values for codemarks, this selectivity can be readily achieved.

[0151] The reporter may provide pre-defined single-shot ~~single-short~~ reporter methods, such as the examples below. When such a reporter method is invoked, it extracts and saves a specified finite block of data to evidence. Such a reporter method includes a procedure (program) for the reporter sequential logic. This drives the controller for L2 cache, and/or the controller for main memory, and/or software to operate storage, particularly software for a memory mapped file. These reporter methods may be implemented as deep-level resources, augmented by system software to operate a file in storage. In many cases, these reporter

methods operate very compatibly with execution of the central-program. Examples of several reporter methods follow.

[0160] {Copy a specified Call Vector from the Call Stack to Evidence} This reporter method uses the list of pointers to the call stack. This reporter method uses two successive pointers to define a range of addresses. Using ~~Use~~ this range and a preceding reporter method, this copies a call vector to evidence. In some cases, this is largely a deep-level resource.[

[0165] {Repeatedly copy branch-family instructions} For every executed branch-family instruction, copy its detailed description to Evidence. This particularly includes every ~~even~~ destination I-address, plus the direction of the branch, plus the program-counter for this instruction. For more discussion, see elsewhere in this invention concerning "negative symptoms".

[0180] The scribe software operates during execution of the central-program. The scribe software controls the means to transfer data, from an evidence buffer in main memory to an evidence file in storage. More completely, the scribe supports the following functions: (i) the sentry, reporter or central-program can activate the scribe; (ii) the scribe can initialize an evidence file in storage; (iii) the scribe can transfer data from an evidence buffer in main memory to an evidence file in storage; (iv) scribe transfers can be done steadily, while data is written into a circular buffer. Thus ~~so that~~ newer evidence can be transferred into the evidence buffer by the sentry, while almost simultaneously older evidence is transferred out to an evidence file in storage; (vi) the evidence buffer can be "flushed" to the evidence file in one shot; (vii) the scribe and reporter together transfer evidence from an evidence buffer in L2 cache to an evidence file in storage; and (viii) the scribe can close an evidence file in storage.

[0181] In one arrangement, the scribe includes scribe sequential logic and scribe memory. This is programmed with a method for each scribe function. The scribe

sequential logic interacts with the following elements: the software resources for Memory Mapped Function (MMF); the deep-level resources for a circular buffer in main memory; and the reporter. The scribe functions correspond closely to functions provided by these three elements. This permits the scribe sequential logic and scribe memory to be especially simple. In one arrangement the scribe is implemented using an extremely small co-processor and memory. This enables additional scribe methods for additional scribe functions. Thus a specialized expert can add more scribe functions. Another arrangement uses sequential gate logic that directly implements a fixed finite state machine. This supports pre-defined scribe functions, but this may restrict additional scribe functions.

[0255] Remote debugging on a Client Computer: Some software is designed to operate on a "client" computer, with a User who is not a software developer. When the user observes a problem ~~probe~~, the user reports it to a software developer. Often, the user provides inadequate evidence of the problem, and it is difficult for the developer to replicate the problem.